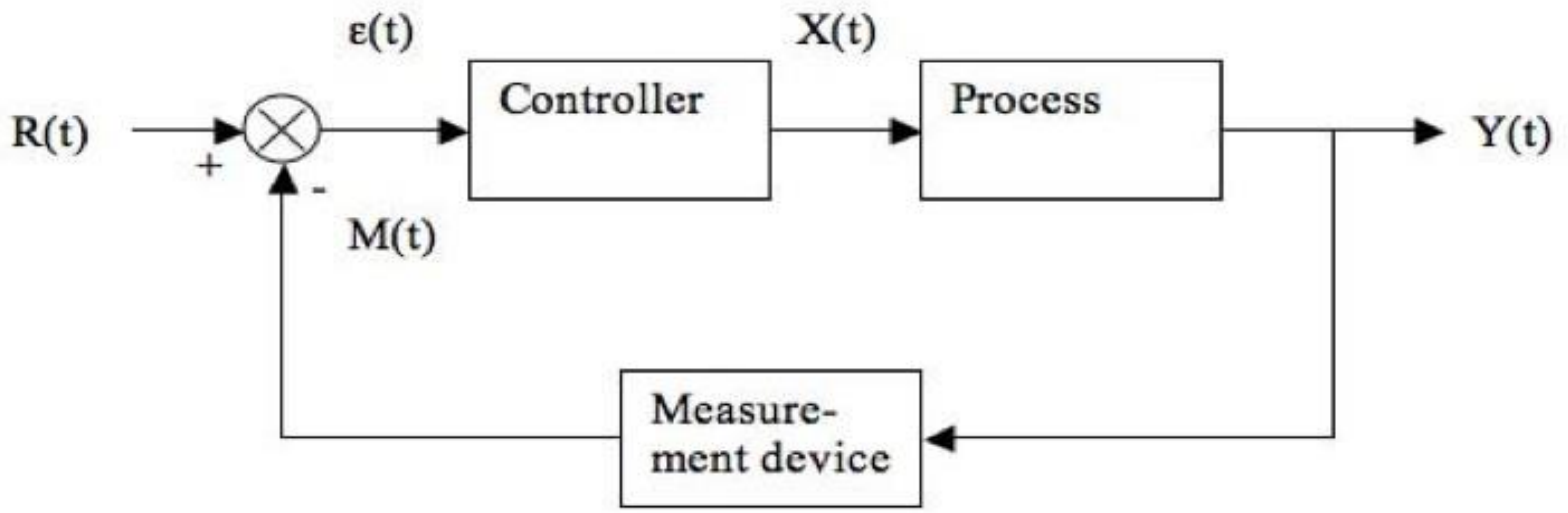
АДАПТИВНИ СОФТУЕРНИ СИСТЕМИ

Software for process control



- Possible software component implementations
  - Controller
    - Sum device may or may not be part of the controller
  - Measurement device (possibly a sensor driver)

What about the process component?

- Physical process
  - Embedded software system
- Software itself
  - Autonomous (dynamic) system

Автономни софтуерни системи

- Autonomic Computing [IBM, 2003]
  - Компютърна среда, която може да се самоуправлява и динамично да се променя според изискванията и целите на бизнеса. Самоуправляващите се среди може да извършват автономни дейности, без нуждата от намеса на ИТ специалисти.

Свойства на автономните системи

- Самоконфигуриране (Self-configuring)
- Самолекуване (Self-healing)
- Самооптимизиране (Self-optimizing)
- Самозащита (Self-protecting)

Самоконфигуриране

- Самоконфигурирането
  - Самоконфигуриращите се компоненти се променят динамично, вследствие на промени в средата с която взаимодействат (потребители и други системи)
- Сред възможните промени са
  - Свързване на нови компоненти
  - Изключване на същесвтуващи компоненти

Самолекуване

- Системата може да открива и диагностицира проблеми, и сама реагира на тях
- Самолекуващите се компоненти откриват дефекти и извършват действия като например
  - Променят собственото си състояние
  - Променят други компоненти, с които взаимодействат

Самооптимизиране

- Системите или техни компоненти може да наблюдават и автоматично да променят ресурси, за да отговорят на нуждите на потребителите и бизнеса.
- Някои възможни промени са например
  - Преразпределяне на ресурси, за да се подобри коефициента на употребата им
  - Подсигуряване на времето за изпълнение на специфични бизнес транзакции
- Текущата практика е оптимизацията да се извършва от операционните системи

Самозащита

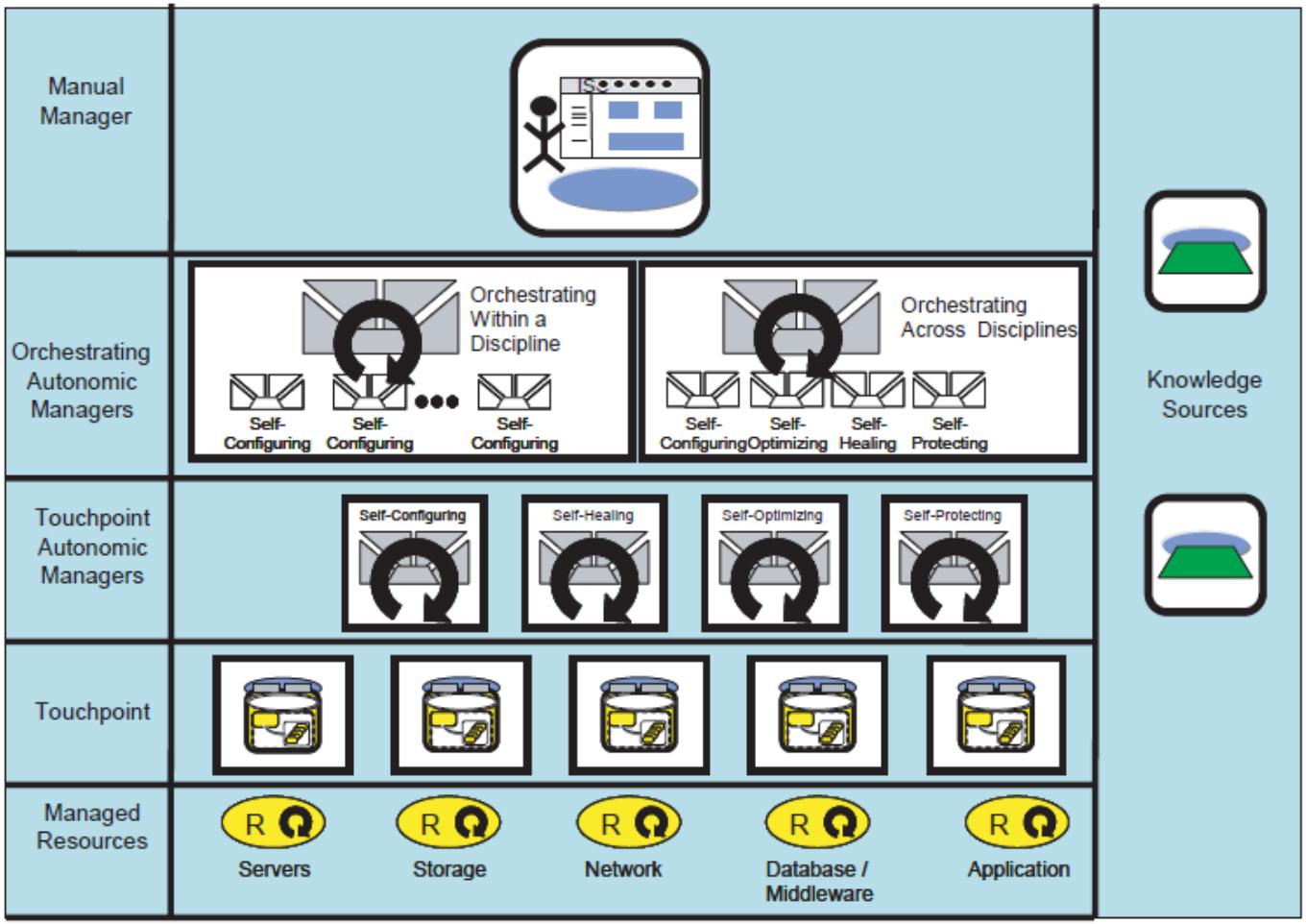
- Самозащитата означава, че компонентите може да предвиждат, откриват, идентифицират заплахи и да предприемат съответните действия за да намалят уязвимостта си.
- Това може да се прави поне за най-известните заплахи към сигурността, които включват
  - Неоторизиран достъп и употреба
  - Вируси и троянски коне
  - Атаки от тип “denial-of-service”

Примери за действия на автономни софтуерни системи

- Самоконфигуриране
  - Инсталация на липсващ софтуер
- Самолекуване
  - Промяна на конфигурация, например локация на някакъв компонент, за да може той да бъде намерен
- Самооптимизиране
  - Промяна на натоварването в следствие на увеличаване или намаляване на капацитета на компонент/ресурс
- Самозащита
  - Извеждане на определени ресурси *offline* ако се бъде засечен опит за проникване

АРХИТЕКТУРА НА АВТОНОМНИ СОФТУЕРНИ СИСТЕМИ

Софтуерно решение



Source: Kephart, Chess, The vision of autonomic computing, 2001

Архитектурата се изгражда на пет нива

- Managed Resources (MR)
- Touch-Points (TP)
- Touch-Point Autonomic Managers (TPAM)
- Orchestrating Autonomic Managers (OAM)
- Manual Manager (MM)

Managed Resources (MR)

- Хардуерен или софтуерен компонент, притежаващ интерфейс, посредством който може да се управлява.
  - Server
  - Storage unit
  - Database
  - Application server
  - Service
  - Network router
  - Personal computer
  - Application and etc.
- MR може да има вградена собствена управляваща архитектура

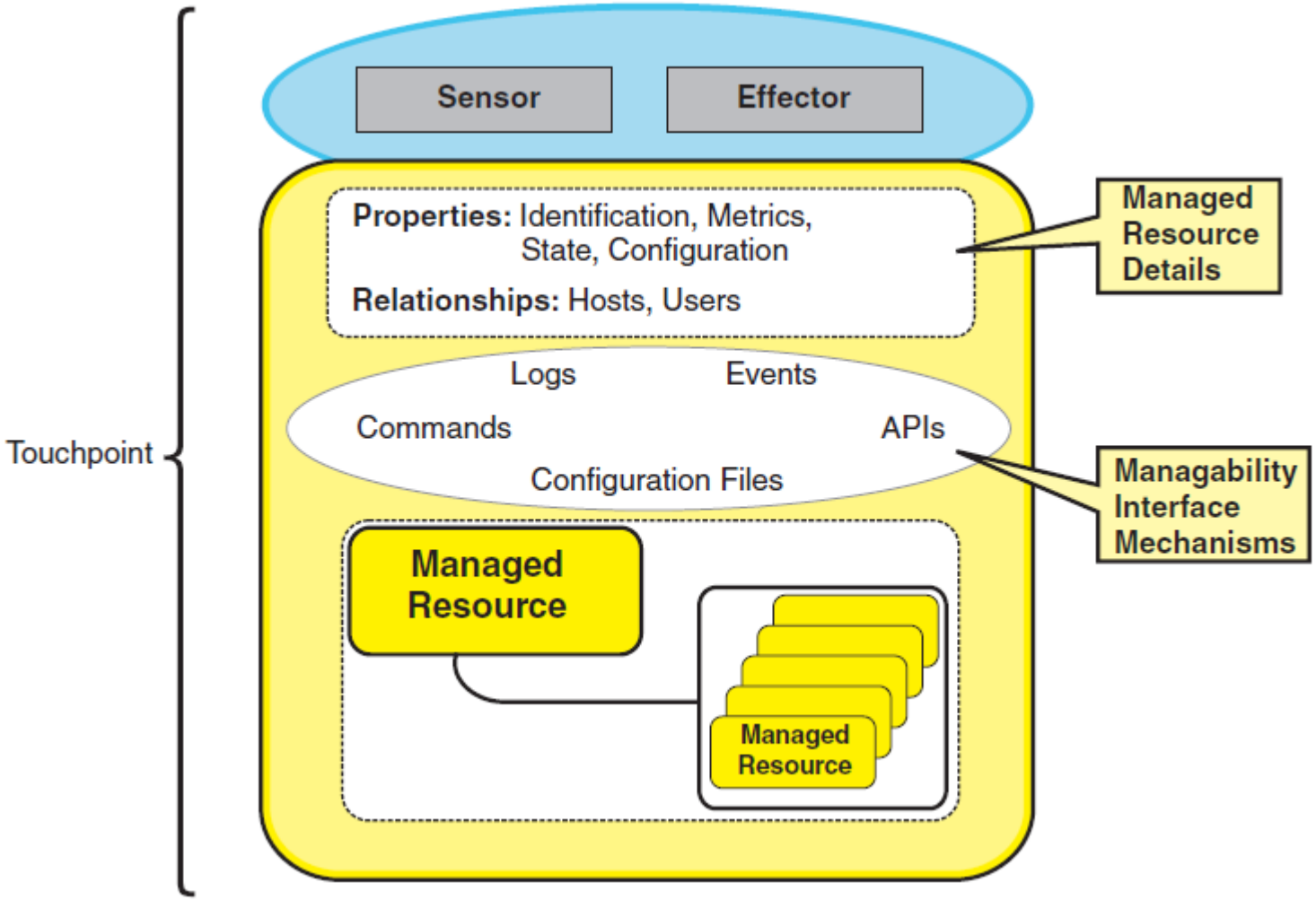
Touch-points

- Модули, в които са реализирани сензор и ефектор, за даден MR
  - сензорите и ефекторите може да се разглеждат като аналог на датчик и управляващ межанизъм
- Осигуряват управляващ интерфейс за достъп и настройка на разпределени MR

Примерни Touchpoint механизми

- Log files
- Events
- Commands
- Application programming interfaces
- Configuration files

Архитектура на Touch-point



Source: http://www.ibm.com/developerworks/autonomic/library/ac-edge5/

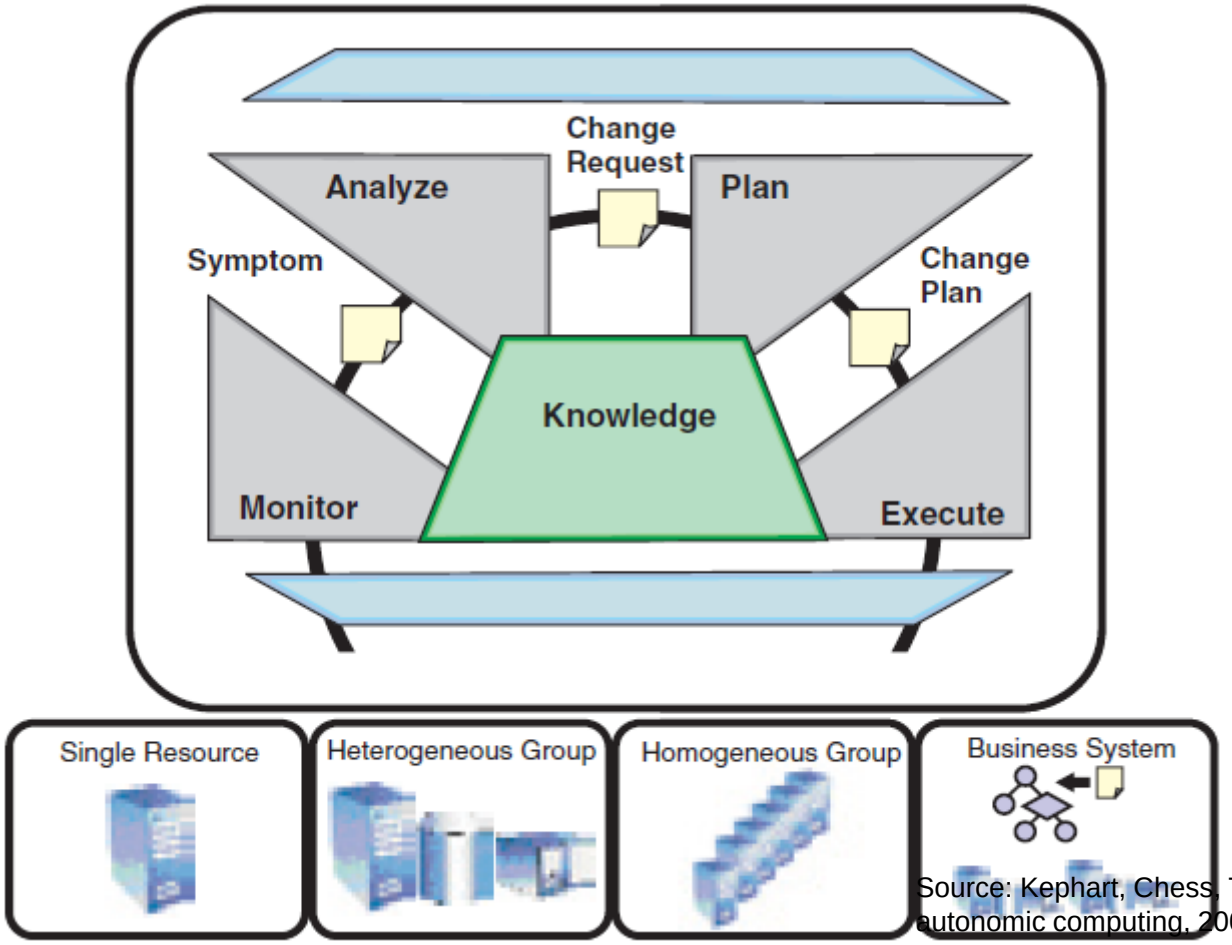
Сензори и ефектори

- Съставни части на сензора
  - Свойства, които предоставят информация за текущото състояние на MR и са достъпни чрез типични “get” методи
  - Събития (events, messages, notifications), които настъпват, когато състоянието на MR се променя
- Съставни части на ефектора
  - Операции (методи), които позволяват да се променя състоянието на MR
  - Операции, реализирани в TPAM, които позволяват на MR да прави заявки към своя TPAM

Touch-point autonomic managers

- Работят директно с MR чрез техните TP
- Реализират специфични интелигиентни обратни връзки (*control loops*)
- Използват се определени политики за управление на поведението на интелигиентните обратни връзки

Autonomic manager control loop



Source: Kephart, Chess. The vision of autonomic computing, 2001

Orchestrating autonomic managers

- TPAM управляват само ресурсите (MR), за които пряко отговарят
- OAM координират действията на TPAM
  - Пример за OAM
    - Workload manager
    - Load balancer

Manual manager

- Supervise the work of the system
- Should take an action in case of exceptions

АРХИТЕКТУРНА АДАПТАЦИЯ И ДИНАМИЧНИ СОФТУЕРНИ АРХИТЕКТУРИ

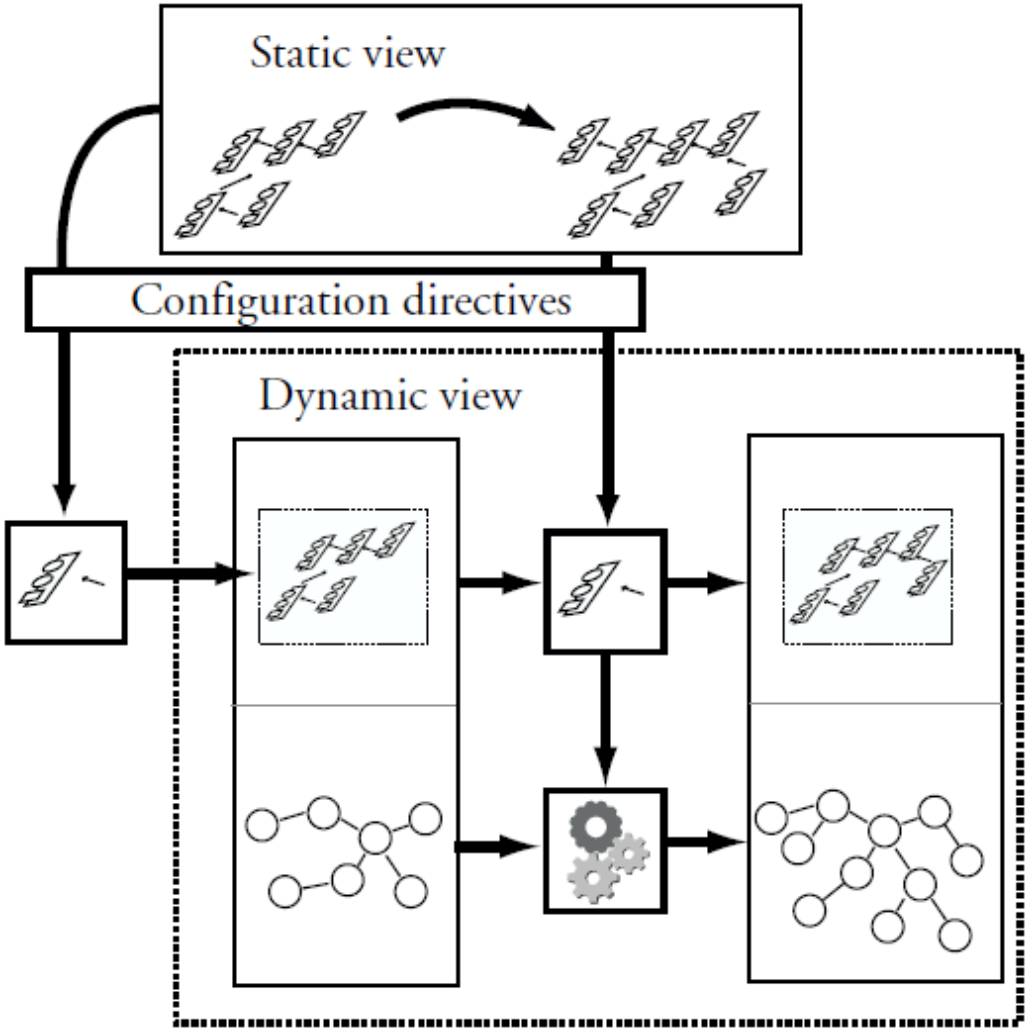
Архитектурна адаптация

- Софтуерна архитектура
  - Съвкупност от структури, които показват абстрактни елементи и връзките между тях
- Модулни структури
- Да си представим, че модулите са елементи не само по време на дизайна на архитектурата, но и по време на изпълнение
  - Ако конфигурацията на модулите по време на изпълнение се променя според някакви правила, говорим за динамични софтуерни архитектури

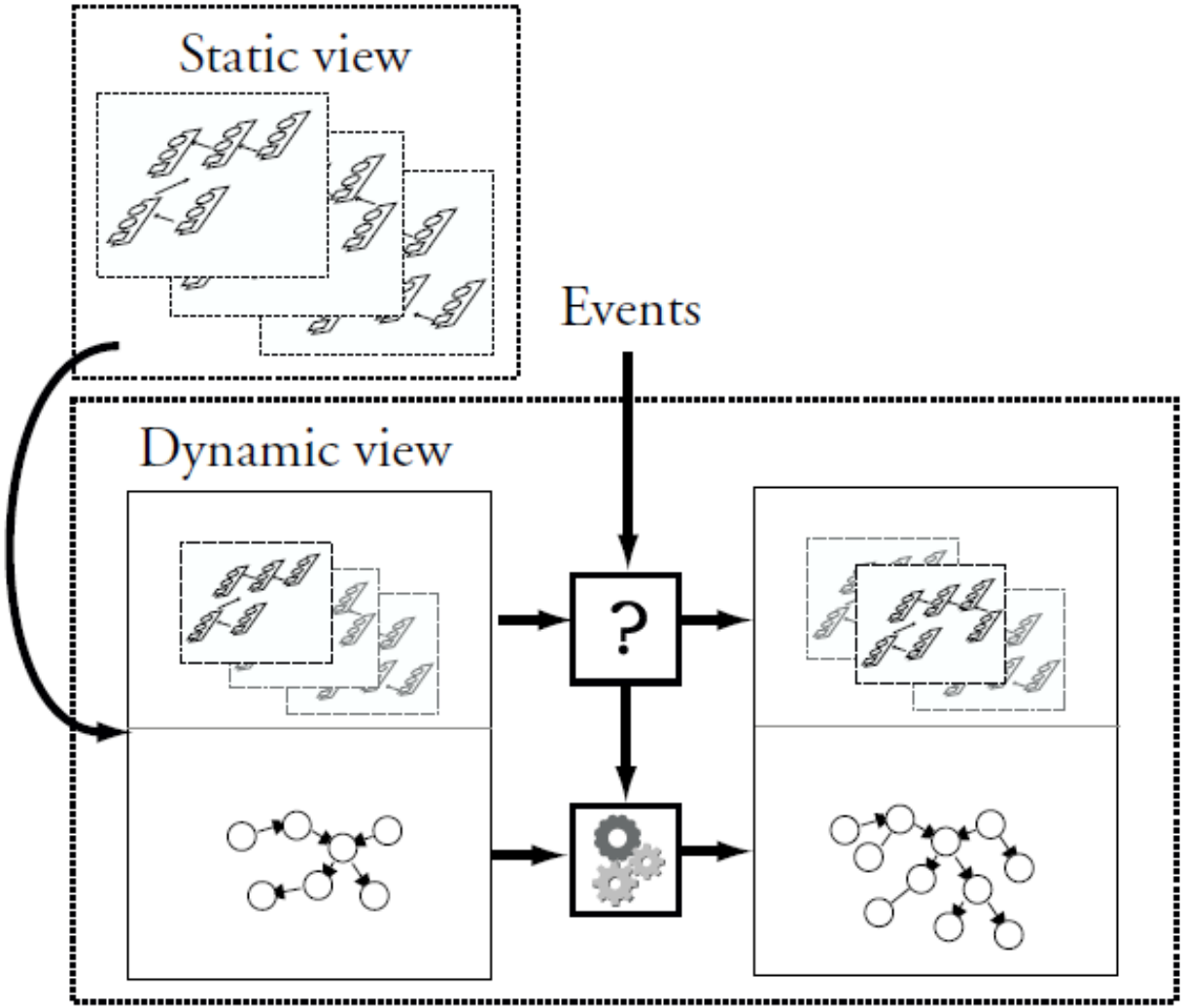
Видове динамизъм в софтуерните архитектури

- Конструктивен динамизъм
  - Възможните причини за промяната на системата, както и самите промени са предварително (твърдо) зададени
- Адаптивен динамизъм
  - В системата е имплементиран механизъм, който да отговаря на промени (събития - events) в околната среда (промени в изискванията, промени в други системи, с които си взаимодейства)
  - Списъкът със събитията може да бъде или да не бъде краен, а списъкът с промените е краен
- Интелигиентен динамизъм
  - Системата сама взема решение как да реагира на промените

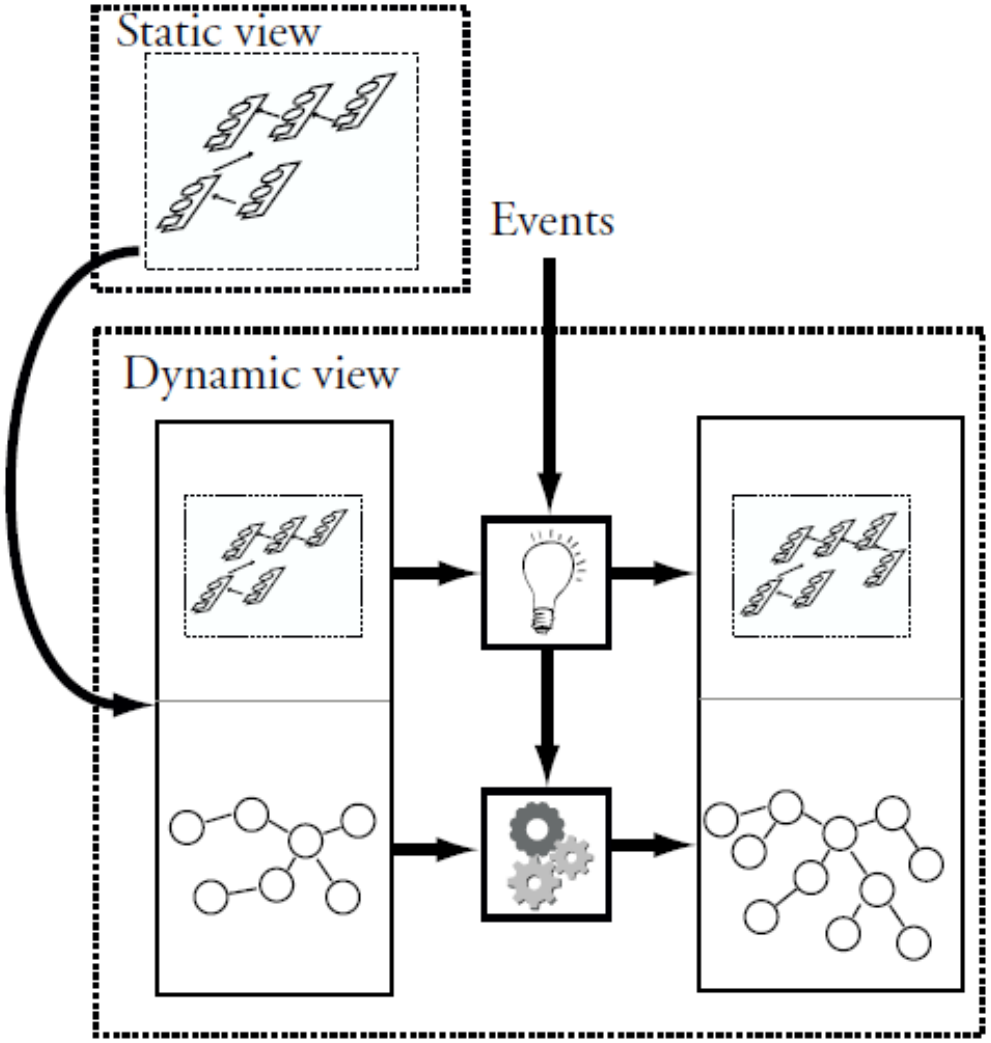
Конструктивен динамизъм



Адаптивен динамизъм



Интелигиентен динамизъм



PRODUCT LINES

Продуктови/Поточни линии

Software product line (SPL)

Дефиниция:

Софтуерната продуктова линия е набор от софтуерни системи, които имат общи функционалности (features), които удовлетворяват специфичните нужди на определен пазарен сегмент и са разработени от набор основни ресурси според предварително предначертан план

- Продуктовата линия може да се разглежда като реализация на:
  - Софтуерно инженерство, ориентирано към специфична проблемна област
  - Динамизъм по време на проектирането

Проблемни области и продуктови линии

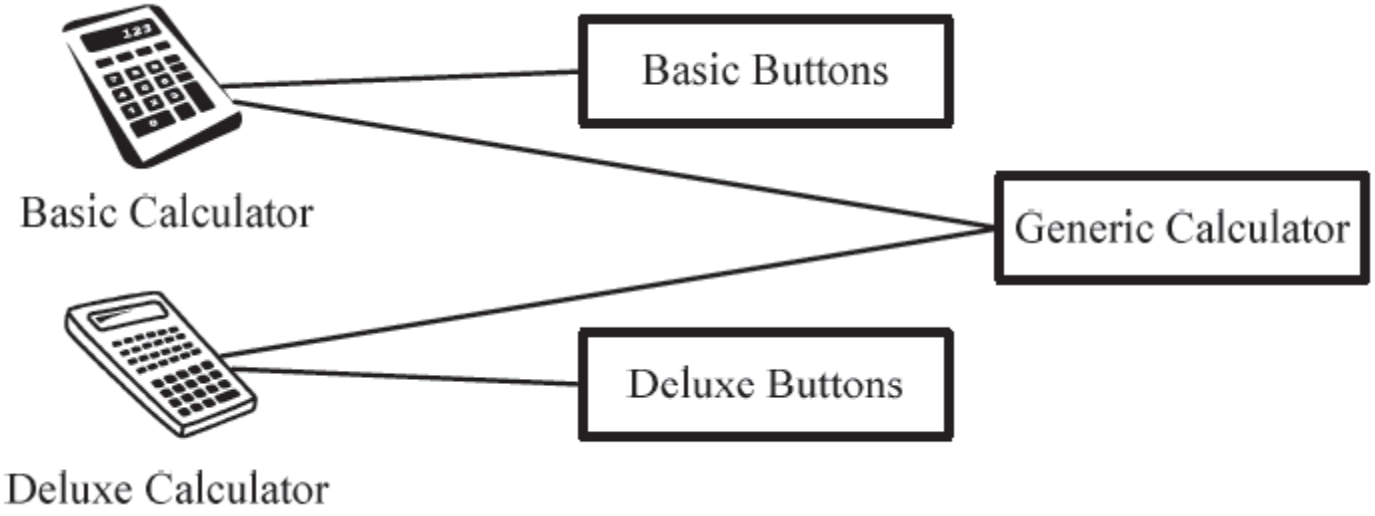
Продуктовите линии представят решения на задачите в проблемните области

- Проблемна област
  - Битова електроника
  - Авиация
  - Компилатори
  - Видеоигри
- Product Line
  - Sony WEGA TVs
  - Фамилия Boeing 747
  - Серията компилатори GNU
  - Серия игри, разработени над един и същи engine

Приложение

- Продуктовите линии намират успешно приложение в следните области:
  - Мобилни телефони
  - Системи за управление на технологични процеси
  - Транспортни системи
  - Финансови системи
  - Битова електроника

Пример

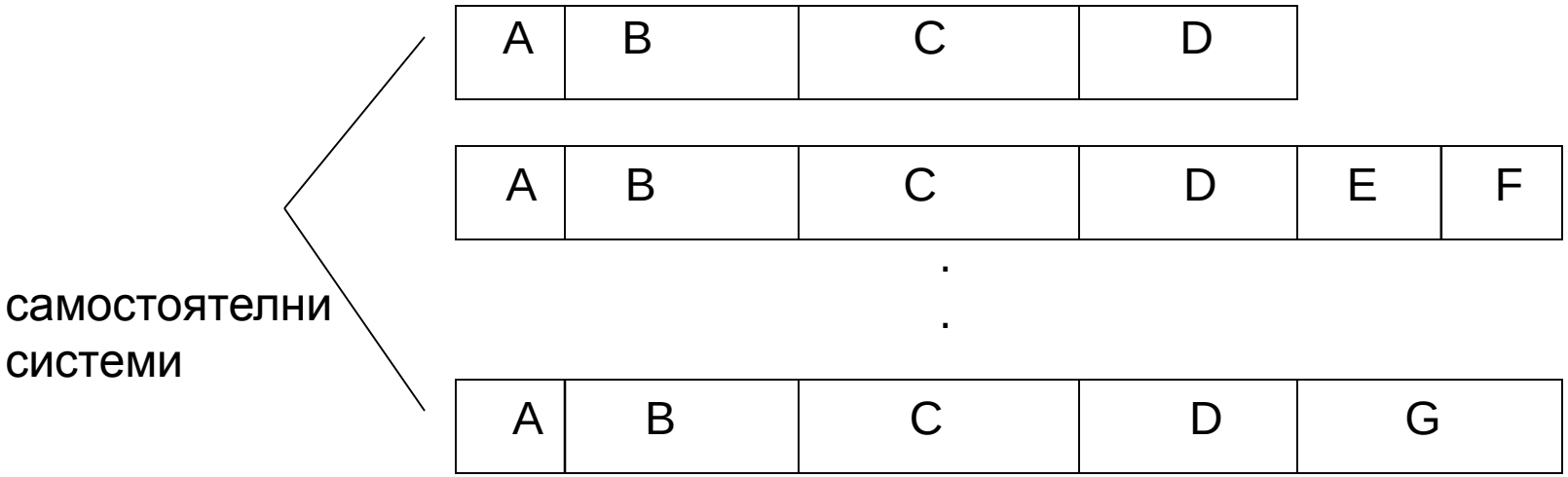

Basic Buttons
Basic Calculator
Generic Calculator
Deluxe Buttons
Deluxe Calculator

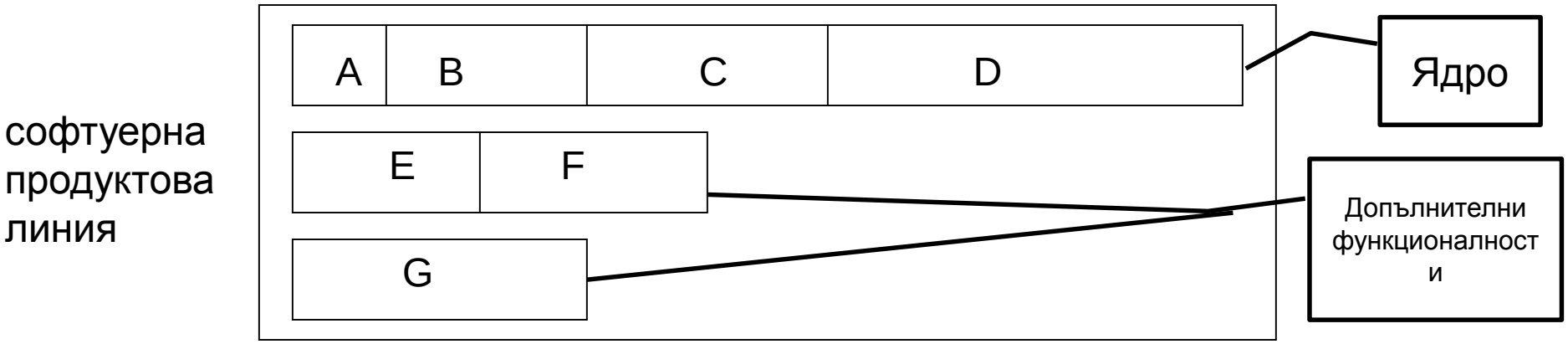
Разработка чрез продуктови линии (ПЛ)

- ПЛ са аналог на поточните линии при производството на хардуер от унифицирани елементи.
- Широко се използват в областта на вградените системи, където изделята представляват комбинация от софтуер и хардуер.
- ПЛ е добър подход, когато дадена организация разработва набор продукти, които имат много общи функционалности.
- Често се случва дадена организация да цели да овладее голям пазар и всеки продукт в ПЛ е ориентиран към специфичен пазарен сегмент.

Продуктови линии и самостоятелни системи

- Когато някоя от основните функционалности в ПЛ се промени, тази промяна се отразява във всички производни системи
- ПЛ може да еволюира и тогава всички системи в нея също еволюират
- Самостоятелните системи нямат горните предимства



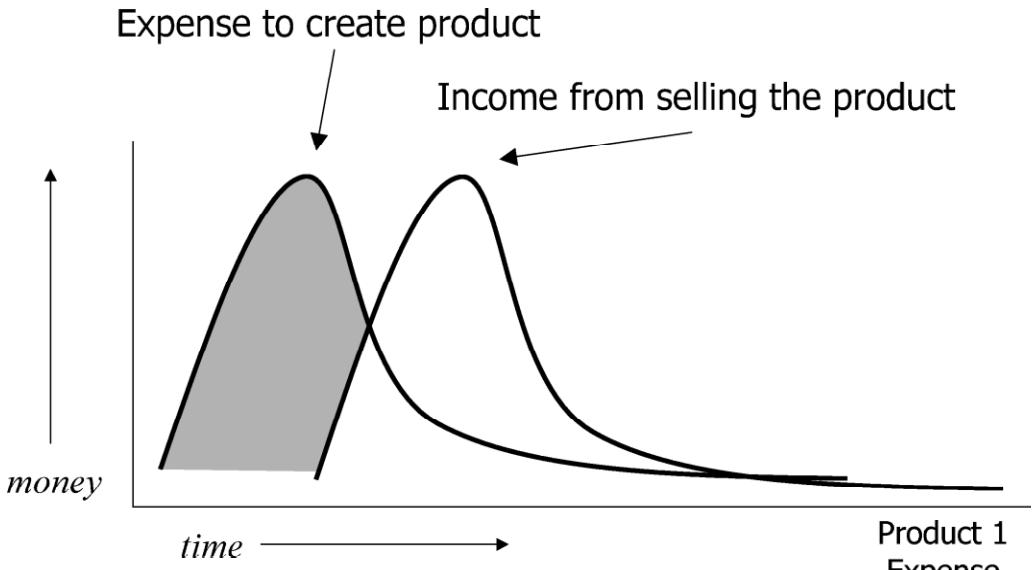


Продуктовите линии са подходящи ако например компонент “C” се наложи да бъде променен. Или пък нов компонент “H” се добави към всяка система

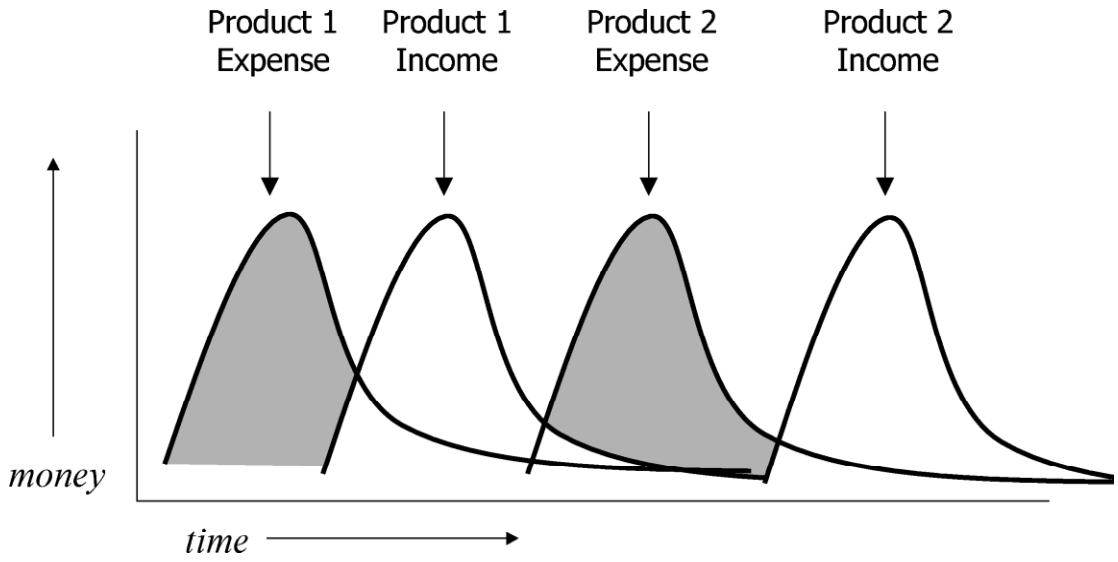
Ползи от продуктовите линии

- Увеличена производителност
- Разходите за разработка и поддръжка на се разпределят между всички продукти в ПЛ
- Крайната цена и време за разработка на нов продукт се понижава
- Дефекти в основнатите функционалности на ПЛ се отстраняват веднъж за всички продукти

Бизнес мотивация

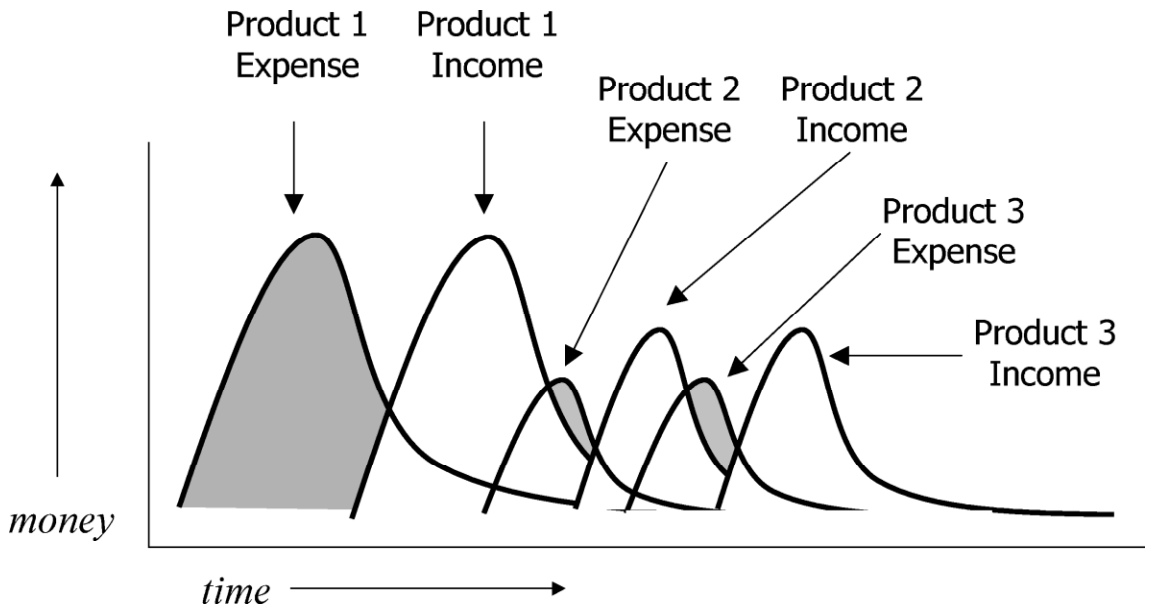


Традиционна
софтуерна разработка



Бизнес мотивация

Разработка базирана на ПЛ



Производство на продукт

- Под продукт на ПЛ разбираме конкретна софтуерна система
- Продуктът представлява своеобразен екземпляр (инстанция) на ПЛ. Това става на две стъпки:
  - Селекция: отстраняват се ненужните ***елементи*** и се уточняват възможните ***вариации***
  - Разширяване: добавят се допълнителни елементи (вероятно разработени от нулата)
- Селекцията е основна стъпка. Трудността идва от това как да подберем основните елементи на ПЛ.
  - Възможните решения се основават на ключови думи, атрибути и т.н.
- В момента най-често използвания подход се базира на функционалности (features).

Функционалност (Feature)

- Може да се разглеждат като абстракция на характерни понятия от терминологията на дадена проблемна област.
- Може да се използват като средство за комуникация между заинтересованите лица
- Пример:
  - “*forward*”, “*reply*” и “*reply all*” може да се разглеждат като *features* на *email* клиент

Пример
(адаптиран от хардуерното производство)

- За да се получи автомобил от „ПЛ“ за автомобили може да се избира от следните „функционалности“: въздушна възглавница (за безопасност), ABS, CD player, комби, двигател1500cc, турбокомпресор и т.н.
- Въпреки избраните функционалности, за да се произведе продукта (автомобил) може да се наложи да се уточнят определени вариации, например : брой въздушни възглавници.

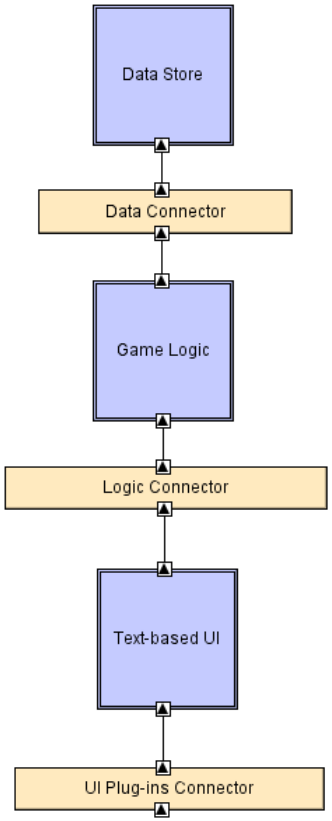
Пример
(софтуерни системи и ПЛ)

- В ПЛ за банкови системи може да се избира от следните функционалности: внасяне, теглене, заем, връщане на заем, обмен на валута и т.н.
- Според избраните функционалности, може да се получат различни банкови системи. Подобно на предишния пример, може да се наложи да се уточнят някои вариации, като например: какви валути да се обменят.

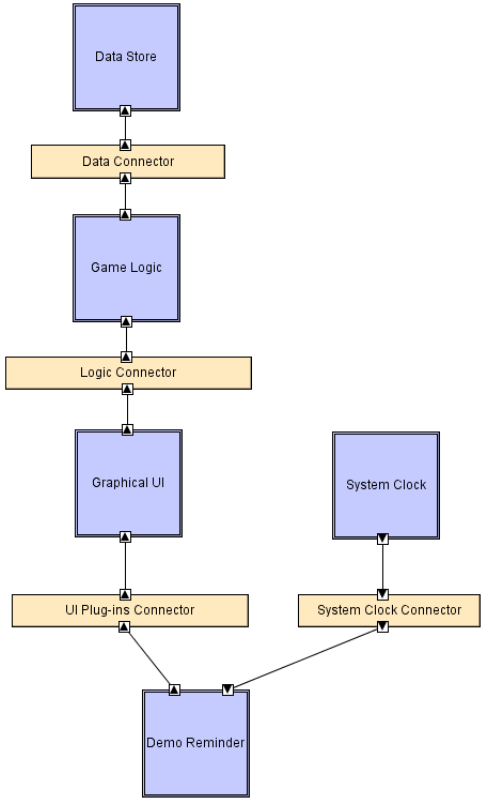
Пример

- Игра симулатор – Lunar Lander
- Има три версии
  - Lite – версия с текстов интерфейс
  - Pro – пълна версия с графичен интерфейс
  - Demo – ограничена във времето за ползване пълна версия

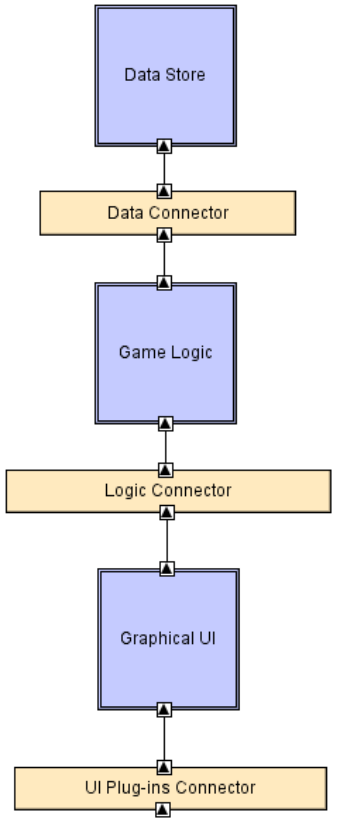
Lunar Lander Product Line



“Lite”



“Demo”



“Pro”

Source: Taylor, R., N. Medvidovic and E. Dashofy, Software Architecture: Foundations, Theory, and Practice, Wiley 2009

Таблица на компонентите в продуктовата линия

Ако има голямо количество общи компоненти в различните версии е разумно да се използва ПЛ.

	Data Store	Data Store Connector	Game Logic	Game Logic Connector	Text-based UI	UI Plug-ins Connector	Graphical UI	System Clock	System Clock Connector	Demo Reminder
Lite	X	X	X	X	X	X				
Demo	X	X	X	X	X	X	X	X	X	X
Pro	X	X	X	X		X	X			

Source: Taylor, R., N. Medvidovic and E. Dashofy, Software Architecture: Foundations, Theory, and Practice, Wiley 2009

Групиране на компонентите във функционалности

	Data Store	Data Store Connector	Game Logic	Game Logic Connector	Text-based UI	UI Plug-ins Connector	Graphical UI	System Clock	System Clock Connector	Demo Reminder
Core Elements	X	X	X	X		X				
Text UI					X					
Graphical UI							X			
Time Limited								X	X	X

Source: Taylor, R., N. Medvidovic and E. Dashofy, Software Architecture: Foundations, Theory, and Practice, Wiley 2009

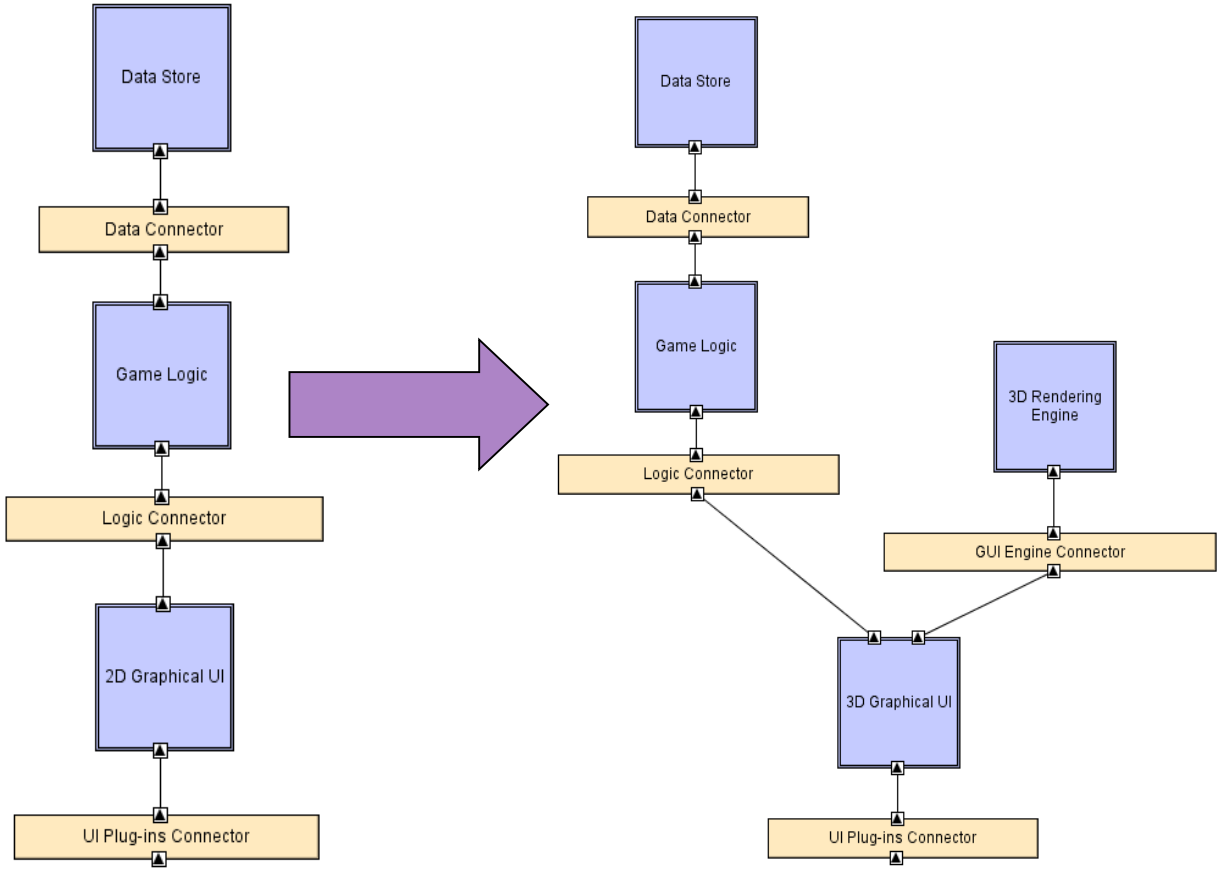
Създаване на продукти

	Core Elements	Text UI	Graphical UI	Time Limited
Lunar Lander Lite	X	X		
Lunar Lander Demo	X		X	X
Lunar Lander Pro	X		X	

Source: Taylor, R., N. Medvidovic and E. Dashofy, Software Architecture: Foundations, Theory, and Practice, Wiley 2009

Product Lines for Evolution

- Продуктите в ПЛ може да са резултат и на промени във времето.
  - Така например може да се добави нов продукт към Lunar Lander, който да има 3D интерфейс.



Source: Taylor, R., N. Medvidovic and E. Dashofy, Software Architecture: Foundations, Theory, and Practice, Wiley 2009